1945: John von Neumann
- Wrote a report on the stored program concept, known as the First Draft of a Report on EDVAC
- also Alan Turing... Konrad Zuse... Eckert & Mauchly...

The basic structure proposed in the draft became known as the “von Neumann machine” (or model).
- a memory, containing instructions and data
- a processing unit, for performing arithmetic and logical operations
- a control unit, for interpreting instructions
The von Neumann Machine

Abstraction of von Neumann Architecture

- **MEMORY**: stores both program instructions and data
  - MAR: Memory Address Register
  - MDR: Memory Data Register

- **PROCESSING UNIT**: decodes current instruction, manages processing unit to carry out instruction
  - ALU: Arithmetic Logic Unit
  - TEMP: Temporary Register

- **INPUT**
  - Keyboard
  - Mouse
  - Scanner
  - Card reader
  - Disk

- **OUTPUT**
  - Monitor
  - Printer
  - LED
  - Disk

- **CONTROL UNIT**
  - PC: Program Counter
  - IR: Instruction Register

**program counter**: points to the next instruction to be fetched

**instruction register**: stores current instruction
Totally dominate laptop/desktop/server market

Evolutionary design
- Backwards compatible up until 8086, introduced in 1978
- Added more features as time goes on

Complex instruction set computer (CISC)
- Many different instructions with many different formats
  ■ But, only small subset encountered with Linux programs
- Hard to match performance of Reduced Instruction Set Computers (RISC)
- But, Intel has done just that!
  ■ In terms of speed. Less so for low power.
### Intel x86 History

<table>
<thead>
<tr>
<th>Name</th>
<th>Date</th>
<th>Transistors</th>
<th>MHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>8086</td>
<td>1978</td>
<td>29K</td>
<td>5-10</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>First 16-bit processor. Basis for IBM PC &amp; DOS</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1MB address space</td>
<td></td>
</tr>
<tr>
<td>80386</td>
<td>1985</td>
<td>275K</td>
<td>16-33</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>First 32 bit processor, referred to as IA32</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added “flat addressing”</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Capable of running Unix</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>32-bit Linux/gcc uses no instructions introduced in later models</td>
<td></td>
</tr>
<tr>
<td>Pentium 4F</td>
<td>2004</td>
<td>125M</td>
<td>2800-3800</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>First 64-bit processor, referred to as x86-64</td>
<td></td>
</tr>
<tr>
<td>Core i7</td>
<td>2008</td>
<td>731M</td>
<td>2667-3333</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Our shark machines</td>
<td></td>
</tr>
</tbody>
</table>
Intel x86 History

Architectures:
- X86-16
- X86-32/IA32
  - MMX
  - SSE
  - SSE2
  - SSE3
- X86-64 / EM64t
  - SSE4

Processors:
- 8086
- 286
- 386
- 486
- Pentium
- Pentium MMX
- Pentium III
- Pentium 4
- Pentium 4E
- Pentium 4F
- Core 2 Duo
- Core i7

Time arrow pointing from X86-64 to Pentium 4.
Intel x86 History

Intel Attempted Radical Shift from IA32 to IA64
- Totally different architecture (Itanium)
- Executes IA32 code only as legacy
- Performance disappointing

AMD Stepped in with Evolutionary Solution
- x86-64 (now called “AMD64”)

Intel Felt Obligated to Focus on IA64
- Hard to admit mistake or that AMD's approach is better

2004: Intel Announces EM64T extension to IA32
- Extended Memory 64-bit Technology
- Almost identical to x86-64!

All but low-end x86 processors support x86-64
- But, lots of code still runs in 32-bit mode
von Neumann View of x86

CPU
  - PC
  - Registers
  - Condition Codes

Memory
  - Object Code
  - Program Data
  - OS Data
  - Stack

Addresses

Data

Instructions
High-level x86 CPU

Programmer-Visible State

- **PC: Program counter**
  - Address of next instruction
  - Called “EIP” (IA32) or “RIP” (x86-64)

- **Register file**
  - Heavily used program data

- **Condition codes**
  - Store status information about most recent arithmetic operation
  - Used for conditional branching
IA32 Integer Registers

16-bit virtual registers (backwards compatibility)

%eax  %ax  %ah  %al
%ecx  %cx  %ch  %cl
%edx  %dx  %dh  %dl
%ebx  %bx  %bh  %bl
%esi  %si  
%edi  %di  
%esp  %sp  
%ebp  %bp  

General purpose:

CPU

PC

Registers

Condition Codes
### x86-64 Integer Registers

<table>
<thead>
<tr>
<th>%rax</th>
<th>%eax</th>
<th>%r8</th>
<th>%r8d</th>
</tr>
</thead>
<tbody>
<tr>
<td>%rbx</td>
<td>%ebx</td>
<td>%r9</td>
<td>%r9d</td>
</tr>
<tr>
<td>%rcx</td>
<td>%ecx</td>
<td>%r10</td>
<td>%r10d</td>
</tr>
<tr>
<td>%rdx</td>
<td>%edx</td>
<td>%r11</td>
<td>%r11d</td>
</tr>
<tr>
<td>%rsi</td>
<td>%esi</td>
<td>%r12</td>
<td>%r12d</td>
</tr>
<tr>
<td>%rdi</td>
<td>%edi</td>
<td>%r13</td>
<td>%r13d</td>
</tr>
<tr>
<td>%rsp</td>
<td>%esp</td>
<td>%r14</td>
<td>%r14d</td>
</tr>
<tr>
<td>%rbp</td>
<td>%ebp</td>
<td>%r15</td>
<td>%r15d</td>
</tr>
</tbody>
</table>

- Extend existing registers. Add 8 new ones.
- Make %ebp/%rbp general purpose
Memory
- Byte addressable array
- Code, user data, (some) OS data
- Includes stack used to support procedures
But the hardware only "understands" binary representations.
int imax(int first, int second) {
    if ( first >= second )
        return first;
    return second;
}

 gcc -O0 -c -m64 -std=c99 imax.c

But who wants to program in binary?
Programming the Machine

Solution:
translate high-level language code into intermediate-level code which is more human-friendly,
then translate that "assembly" code into the machine's language.

```c
int imax(int first, int second) {
    if ( first >= second )
        return first;
    return second;
}
```

```assembly
... imax:
    pushq  %rbp
    movq  %rsp, %rbp
    subq  $8, %rsp
    movl  %edi, -4(%rbp)
    movl  %esi, -8(%rbp)
    movl  -4(%rbp), %eax
    cmpl  -8(%rbp), %eax
    jl    .L4
    movl  -4(%rbp), %eax
    jmp   .L5
  .L4:
    movl  -8(%rbp), %eax
  .L5:
    popq  %rbp
    ret
...`
```
### Explanation: moving data

<table>
<thead>
<tr>
<th>Source</th>
<th>Dest</th>
<th>Src,Dest</th>
<th>C Analog</th>
</tr>
</thead>
<tbody>
<tr>
<td>Imm</td>
<td>Reg</td>
<td>movl $0x4,%eax</td>
<td>tmp = 0x4;</td>
</tr>
<tr>
<td>Mem</td>
<td>Reg</td>
<td>movl $-147,(%rax)</td>
<td>*p = -147;</td>
</tr>
<tr>
<td></td>
<td>Mem</td>
<td>movl %eax,(%rdx)</td>
<td>*p = tmp1;</td>
</tr>
<tr>
<td>Mem</td>
<td>Reg</td>
<td>movl (%rax),%edx</td>
<td>tmp = *p;</td>
</tr>
</tbody>
</table>